Wireless Pers Commun (2009) 48:247–260
DOI 10.1007/s11277-008-9520-3

Partial CRC Checksum of SCTP for Error Control over Wireless Networks

Lin Cui · Seok Joo Koh

Published online: 20 June 2008

Abstract In the Stream Control Transmission Protocol (SCTP), when a portion of a packet is corrupted, the entire packet will be discarded at the receiver side. This may result in degradation of the throughput of SCTP over wireless networks with a high bit error rate. This paper proposes a new error control scheme of SCTP using a partial Cyclic Redundancy Check (CRC) checksum to enhance the throughput performance, in which a new 'checksum chunk' is introduced to effectively identify any corruptions of data chunks contained in the SCTP packet. In the proposed scheme, an SCTP data packet can carry one or more data chunks depending on the channel condition, and the newly defined 'checksum' chunk will contain the partial CRC checksums of the individual data chunks and/or the base header of the packet. By referring to these partial checksums, the receiver can discard only the corrupted data chunks, whereas the other available data chunks can be recovered. Simulation results show that the proposed scheme significantly provides better performance than the standard SCTP in the wireless networks.

Keywords SCTP · Corruption · Checksum · Partial CRC · Wireless networks

1 Introduction

The Stream Control Transmission Protocol (SCTP) [1] is originally designed for transport of Signaling System 7 (SS7) signaling messages over IP networks, and now has been extended as a general-purpose reliable transport layer protocol. Differently from Transport Control Protocol (TCP), the SCTP provides some unique features such as multi-streaming and multi-homing. The multi-streaming feature can be used to alleviate the head-of-line (HoL) blocking effect of TCP, where each stream within the overall data flow can be delivered

L. Cui · S. J. Koh (✉)
Department of Computer Science, Kyungpook National University, Daegu, Korea
e-mail: sjkoh@cs.knu.ac.kr

L. Cui
e-mail: cuilin@cs.knu.ac.kr

to the upper-layer application independently of the other streams [2,3]. The multi-homing feature allows the two endpoints to establish an association using multiple IP addresses. This feature can be used by SCTP endpoints to improve the efficiency of data transmissions by performing the primary path change in the face of a path failure [4–6], or by allowing a sender to transmit data to a multi-homed receiver through different destination addresses simultaneously [7–10].

On the other hand, the SCTP is designed based on the window-based error and congestion control [11]. As done in TCP, a corrupted packet is regarded as a loss and thus induces the retransmission request at the receiver side and the decrease of the congestion window at the sender side. This will cause the further degradation of SCTP throughput over wireless networks with a high bit error rate (BER).

Several works have been done to improve the SCTP performance against corruption. The work in [12] introduces a MAC-Error-Warning (MEW) method with a new type of packet generated at Medium Access Control (MAC) layer. The MEW method is similar to the Automatic Request (ARQ) mechanism [13], but a special MAC packet is generated before a data packet is discarded, which contains the detailed information (including the stream ID and the sequence number) to notify the source of the failed transmission. Arriving at the transport layer, the MEW packet will be analyzed by the SCTP sender to trigger a fast retransmission of the data chunk without degrading the congestion window. The work in [14] utilizes the Explicit Congestion Notification (ECN) [15], which will be marked by an ECN-capable router in the network. Accordingly, a packet loss without an ECN message will be regarded as a non-congestion error, and the sender will execute a simple loss-recovery procedure without applying the normal congestion control mechanism.

The same issues have also been addressed in a lot of literatures on TCP [16–21], among which as a feasible solution against link-level errors, the Forward Error Correction (FEC) scheme recently has attracted much attention and has been suggested for satellite channel [22]. The drawback of FEC is to consume some extra bandwidth to transmit the redundant information. On the other hand, the study in [23] shows that there is an optimal amount of redundancy to be added for performance enhancement.

Moreover, with the rapidly increased usage of wireless devices all over the world, the faster and cheaper link level (e.g., 802.11) mechanisms tend to be simple (e.g., 802.11's ARQ mechanism). This trend results in a high and variable residual erasure rate (e.g., 10–50% erasure rate observed by [24]) that needs to be ultimately handled in the end-to-end manner [16].

This paper deals with the SCTP error control over wireless network. This work is motivated from the observation that an SCTP segment can carry multiple data chunks but only a single overall checksum is used in its common header. Accordingly, on reception of an SCTP segment, if the checksum field indicates a corruption, the entire segment will be discarded even though only a portion of the multiple data chunks is corrupted during transmission. Therefore, if the receiver can recover the available (successfully received) data chunks from a corrupted SCTP segment, then the sender can be able to reduce the retransmission amount and to avoid the unnecessary halving congestion window incurred by corruptions, in particular, under the wireless network with a high BER.

In this paper we propose a partial Cyclic Redundancy Check (CRC) checksum scheme so as to enhance the SCTP error control over wireless networks. In the proposed scheme, each data chunk contained in the SCTP segment can have its own checksum value that is used to detect a corruption on an individual chunk basis. To support this partial CRC checksum scheme, a new 'checksum' chunk is introduced, and also the formats of the existing INIT/INIT-ACK and SACK chunks will be extended. The main intention of the proposed scheme is to reduce the amount of unnecessary retransmissions in addition to avoiding the

unwanted deflation of congestion window, by using a flexible chunk checksum policy over the wireless channels.

In the proposed scheme, the two SCTP endpoints will negotiate each other about whether the partial CRC checksum is supported or not during the association establishment phase. This is done by using a newly defined parameter option contained in the INIT and INIT-ACK chunks. If the partial CRC checksum is supported, the SCTP sender can transmit variable data chunks with the corresponding 'checksum' control chunk to the receiver depending on the channel condition. On the other hand, if the overall checksum in the common header for a segment fails (i.e., a corruption of the SCTP segment), the SCTP receiver will check whether each of the data chunks contained in the segment fails or not. This is done for the receiver by referring to the partial CRC checksum values of the associated 'checksum' chunk. Through this checking process of partial checksums, some of the data chunks in the entire segment may be recovered as available data chunks, if any. Then, the receiver will deliver the successfully received in-order data to the upper layer in time. After that, the receiver will report the associated Transmission Sequence Numbers (TSNs) to the sender via the subsequent SACK chunks so as to indicate which data chunk is corrupted. The fast retransmission is requested, if necessary, without shrinking the congestion window. For this purpose, this paper also suggests an extension of the SACK chunk format.

The core problem is that SCTP packets are often lost within the network due to failed link-layer (L2) CRCs, more often than at the end-node due to failed transport layer (L4) checksum. To let SCTP handle the corrupted packets with L4 checksum, two approaches can be considered. One is to disable the L2 CRC completely; the other is to set a 'skip flag' in each packet's header for L2 CRC mechanism to skip them. The former may take the risk for other transport protocols to deliver some garbage to the application (e.g. TCP, since TCP employs 16-bit 1's complement checksum which is relatively weaker than 32-bit CRC mechanism) and overload transport layer, the later needs a slight revision for L2 CRC mechanism to ignore the special packets with the 'skip flag'.

The rest of this paper is organized as follows. Section 2 describes the SCTP control chunks used to support the partial CRC checksum scheme, in which the formats of the INIT, INIT-ACK, and SCAK chunks are extended and a 'checksum' chunk is newly defined. Section 3 describes the error control mechanism based on the proposed partial checksum scheme. Section 4 shows the experimental simulation results for the proposed scheme using the ns-2 networks simulator. Finally, we conclude this paper in Sect. 5.

2 Extensions of SCTP Chunks for Partial CRC Checksum

The proposed partial CRC checksum scheme is purposed to enhance the throughput performance of SCTP over wireless channels by differentiating a corruption of an individual data chunk from a corruption of the entire SCTP segment. For this purpose, the CRC checksums of individual data chunks will be calculated, in addition to the entire checksum of the SCTP common header. To support the proposed scheme, we need to define a new 'checksum' control chunk. In addition, a new parameter option is defined and contained in the INIT and INIT-ACK chunks, and the SACK chunk format will also be extended.

2.1 Optional Parameter for INIT/INIT-ACK Chunks

To apply the proposed scheme, the two SCTP endpoints need to negotiate the use of the partial CRC checksum scheme. To do this, in the SCTP association establishment phase, the sender

Table 1 Optional parameters for SCTP

Variable parameters	Status	Type value
IPv4 address	Already defined	5
IPv6 address	Already defined	6
Cookie preservative	Already defined	9
Reserved for ECN capable	Already defined	32768 (0×8000)
Host name address	Already defined	11
Supported address types	Already defined	12
Partial checksum chunk	Newly defined	13

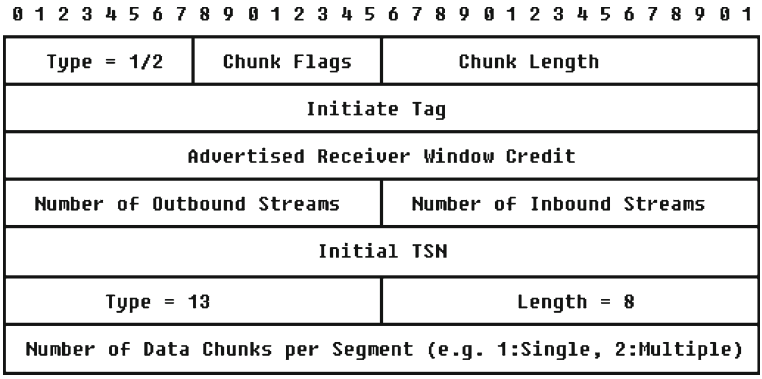

Fig. 1 SCTP INIT/INIT-ACK chunk with the partial checksum option

and the receiver need to confirm whether or not to use the optional partial CRC checksum chunk and how many data chunks should be carried per segment over the association by exchanging the SCTP INIT and INIT-ACK chunks with the associated optional parameter.

Table 1 lists all the optional parameters supported by the standard SCTP, together with a new parameter for the partial CRC checksum proposed in this paper. In the table, a new option parameter with the type value of '13' is assigned for the proposed partial CRC checksum scheme.

The new optional parameter will be contained in the INIT and INIT-ACK chunks for the proposed checksum scheme. For example, Fig. 1 shows the INIT or INIT-ACK chunk that contains the 'partial checksum' optional parameter. When both of the two SCTP endpoints support this option, the proposed scheme will be enabled in the end-to-end data transmission.

2.2 Checksum Chunk

Once the proposed scheme is enabled between the two endpoints, the SCTP sender will transmit the data packets to the receiver. Each data packet can include one or more data chunks depending on the channel condition, and the newly defined checksum chunk will contain the information on the partial CRC checksums for the packet's header and the respective data chunks contained in the SCTP data packet if multiple data chunks are carried.

In a data packet, the checksum chunk will be inserted with the chunk type of '15', next to the common header. Figure 2 illustrates the format of the proposed checksum chunk, which is designed based on the RFC2960 [1].

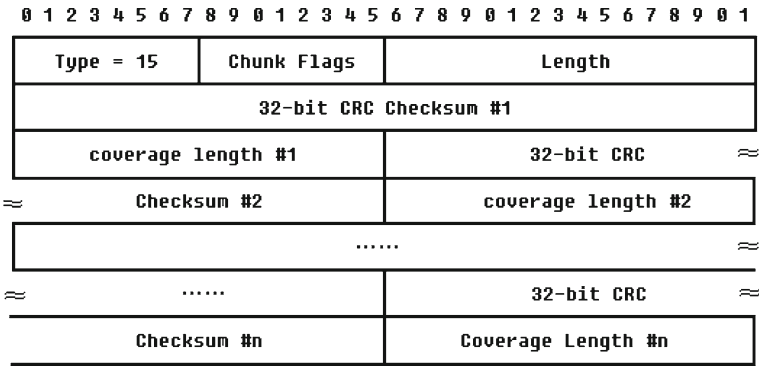

Fig. 2 Checksum chunk for the partial CRC checksum

In the figure, each of the partial checksum items consists of the 4-byte checksum value field (e.g., CRC Checksum #1) and the 2-byte coverage length field (e.g., coverage length #1), which will be a total 6-byte in length. It is noted that the first partial checksum item is responsible for the checksum of a packet portion which covers from the beginning of the packet to the region indicated by the corresponding coverage length in byte.. Likewise, the second partial checksum is responsible for the checksum of the subsequent area of the packet which covers the associated coverage length (in byte) from the end of coverage area of the first partial checksum. Each of the partial CRC checksum items will be configured in this way.

It is noted that the first partial checksum of the checksum chunk is responsible for the header portion that typically consists of the SCTP common header, checksum chunk, other control chunks, if any, as well as the first data chunk's header. The checksum chunk of a segment with only a single data chunk will include a single partial checksum. On the other hand, in the case of the checksum chunk of a segment with multiple data chunks, the respective partial checksums of the individual data chunks will be additionally inserted. For example, the checksum chunk for a segment with a single data chunk will be a 10-byte chunk (4-byte chunk header plus a 6-byte partial checksum item), which is called 'base checksum chunk' in this paper, whereas the checksum chunk of the segment with two data chunks will be a 22-byte chunk, which corresponds to the 10-byte base checksum chunk plus the 12-byte additional checksum items for the two data chunks.

2.3 Extension of SACK Chunk

When the SCTP destination receives the data packet with the checksum chunk from the source, it will first check whether the entire packet is corrupted or not. If the overall checksum fails, the receiver will decide whether or not each of the individual data chunks is corrupted by referring to the checksum chunk. Then, the integral data chunks will be recovered (if any), whereas the corrupted ones shall be notified to the sender. For this purpose, the receiver will respond with the SACK chunk to the sender, which contains the information on the TSNs of the associated data chunks together with the associated timestamp.

Figure 3 shows the format of the extended SACK chunk proposed in this paper. In the figure, the later part of the SACK chunk is added to the existing SACK chunk so as to indicate which data chunks are corrupted.

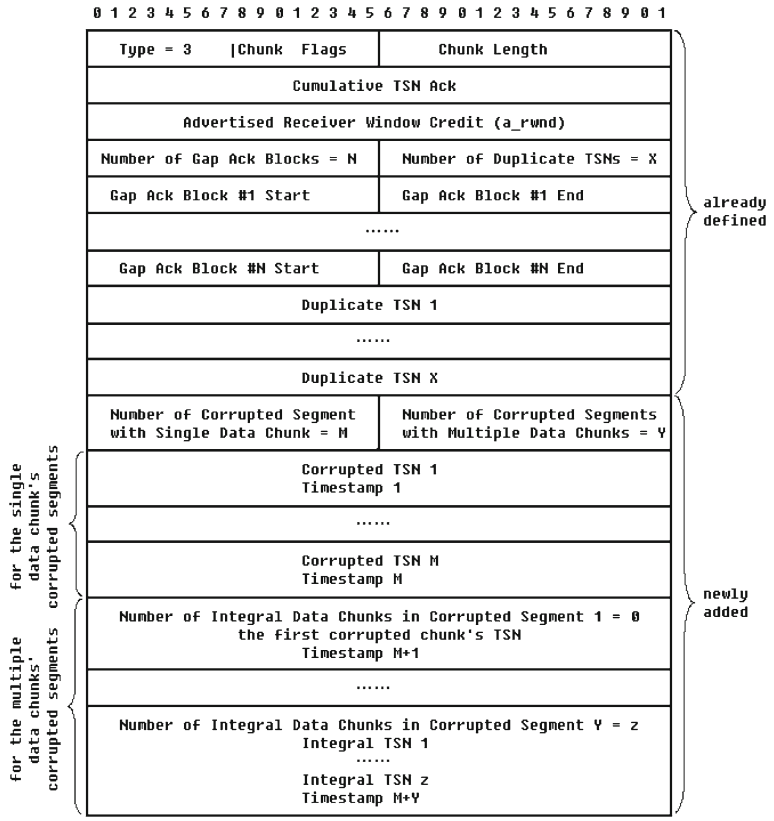

Fig. 3 Format of the extended SACK chunk

In the figure, for each corrupted segment with a 'single' data chunk, an associated corruption item is constructed with a corrupted chunk's TSN as well as a corresponding timestamp (called 'explicit corruption item' in this paper), which is used to explicitly report a corrupted chunk to the sender. On the other hand, the corruption item for the corrupted segment with 'multiple' data chunks, which is called 'implicit corruption item' in this paper, can include one of the following two components: (1) the firstly corrupted data chunk's TSN in case that all the data chunks are corrupted; and (2) the successfully recovered (integral) data chunks' TSNs in case that some individual chunks are recovered (instead of the corrupted ones).

It is noted that on extraction of an implicit corruption item from a SACK chunk, the sender needs to infer which unacknowledged packet has included these chunks and to decide which data chunks need to be retransmitted. Technically, it is easy to identify the corrupted segment as well as the corrupted data chunks, since the sender has only to preserve the contents of all the outstanding packets until they are acknowledged.

3 Error and Congestion Control with CRC Partial Checksums

To apply the proposed scheme, the two SCTP endpoints shall negotiate each other to use the partial CRC scheme through the INIT and INIT-ACK chunks in the association establishment phase. Then, in the data transmission phase, the sender will transmit the data packets

containing one or more data chunks, together with the corresponding checksum chunk proposed in this paper.

In this phase, the sender may adjust the number of data chunks contained in a data packet, depending on the measured packet's corruption rate. For example, when the packet corruption rate decreases down to a pre-configured lower threshold (e.g. 1%), the checksum chunk can not be used any more. Whereas if the packet corruption rate exceeds a pre-specified higher threshold (e.g. 10%), the proposed checksum chunk can be inserted into the packet with multiple data chunks. As stated in Sect. 2.2, the checksum chunk of a single data chunk's segment contains only one partial checksum which indicates whether the packet header is corrupted or not. On the other hand, for a multiple data chunks' segment, each respective data chunk has its own partial checksum, in addition to the first partial checksum for the packet's header.

It is noted that the SCTP sender needs to keep the mapping of the list of data chunks for each outstanding packet in order to identify the corrupted chunks based on the feedback SACK chunks from the receiver.

3.1 Detection of Corrupted Data Chunks by Receiver

On reception of an SCTP segment, the receiver will first verify the integrity of the entire SCTP segment by checking the overall checksum of the common header. In case that the segment is corrupted, the receiver will then verify each partial checksum in turn.

Once checking a partial checksum fails, if it is the first one, the receiver has to discard the whole segment since the header portion may contain some wrong information. On the other hand, if the first partial checksum is proven to be valid (i.e., the information of the packet header is valid), the subsequent partial checksums, if any, will be checked in sequence. For each of the subsequent partial checksums (only for the segment with multiple data chunks), if it is proven to be valid, the corresponding data chunk is available and thus recovered from the corrupted segment. The detailed packet processing procedure at the receiver is illustrated in Fig. 4.

3.2 Generation of SACK Chunks by Receiver

After checking all of the partial checksums, the SCTP receiver will construct a responding SACK chunk immediately, in which the associated TSNs and timestamp will be contained, as shown in Fig. 3.

In particular, if a single data chunk's segment is corrupted, an explicit corruption item will be appended. Otherwise, if the corrupted segment contains multiple data chunks, an implicit corruption item is needed. Herein, the timestamp indicates when the corruption is detected at the receiver side. Therefore, even for the same TSN, a renewed timestamp implies a new corruption event, and thus it requests another prompt retransmission. This allows the sender to retransmit the same chunk multiple times before the retransmission timer expires. By this, the sender can avoid the unwanted timeouts which might be problematic in the conventional SCTP.

3.3 Error Control by Sender

In the standard SCTP, both the lost and corrupted chunks will be retransmitted by the timer-based retransmission or the fast retransmission in the same way. In the proposed

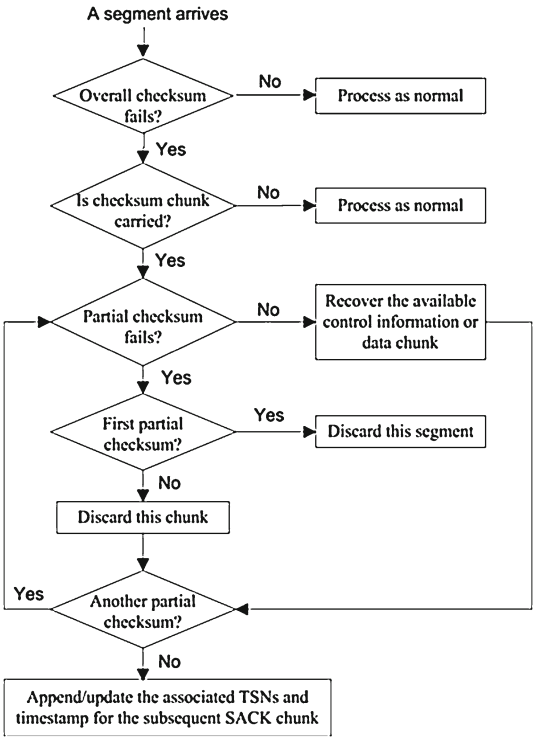

Fig. 4 Detection of corrupted data chunks in a data packet

scheme, however, the corrupted chunk can be processed differently from the loss one, since the corruption itself indicates an explicit retransmission request.

By comparing the record of transmitted data packets with the corruption items enclosed in the feedback SACK chunk, the sender can easily infer whether or not a corruption item reports a new corruption event. In case of a new corruption event, the sender will retransmit the corrupted chunks immediately without deflating its congestion window.

Figure 5 shows the processing of a SACK chunk by the sender. In the figure, when the sender receives a SACK chunk, it first checks whether there are any corruption items. If no, the sender processes it as normal. If any, the sender further determines whether the first item reports a new corruption event. If it does, then the sender records the corrupted TSNs and timestamp for performing the prompt retransmission. Otherwise, the sender simply ignores it and continues to check the next one.

With the help of the corruption items, the sender can adjust its chunk policy based on the measured packet corruption rate. For example, in the network with a higher corruption rate, the sender may contain multiple data chunks with the checksum chunk in the data packet, whereas in the network with a lower corruption rate the data packet may contain only a single data chunk without the checksum chunk. That is, the proposed scheme can be used optionally and selectively together with the standard SCTP. By this, the proposed scheme can significantly improve the throughput performance of SCTP over wireless network with a high BER.

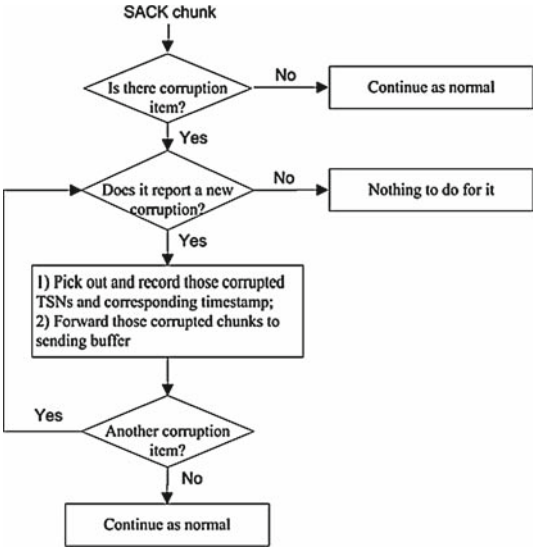

Fig. 5 Processing of the SACK chunks

Fig. 6 Simulation topology

4 Simulation Results

We evaluate the proposed scheme using the ns-2 network simulator (version 2.30) [25] with a simple test topology, as shown in Fig. 6, in which two endpoints communicate through a single path.

In the figure, we assume that the bottleneck wireless link has the bandwidth of 2 Mbps and the end-to-end transmission delay is 35 ms. In each experiment, we perform the file transfer application over 200 s and compare the goodput (Kbps) of the standard SCTP with that of the proposed scheme.

To emulate packet corruptions, we employ Gilbert model [26] over wireless link, which is modified to add a corruption flag in the corrupted packet's header, as shown in (Fig. 7).

In the model, the two states of 'good' and 'bad' are expressed in terms of the average error rates α and β, transition probabilities p and q, and average 'good' period of $t1$ seconds and 'bad' period of $t2$ seconds. If the link is in a 'good' state at present, it will continue to stay in the 'good' state with probability p, or transfer to the 'bad' state with a probability 1-p at the next instance. In the Gilbert model, 'good' state means zero error probability. Also, if the link is in a 'bad' state at the current instance, then it will continue to stay in the 'bad' state with probability q, or transfer to the 'good' state with a probability 1-q at the next instance.

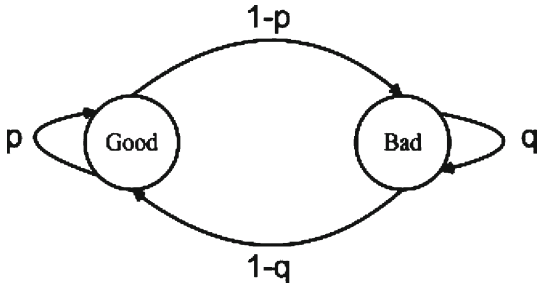

Fig. 7 Two-state Gilbert model for packet corruption

Table 2 Simulation parameters used for Gilbert error model

State	Average period	Transition probability	Packet corruption rate
Good	$t1 = 0.5$ s	$p = 0.5$, $(1 - p) = 0.5$	$\alpha = 0$
Bad	$t2 = 0.5$ s	$(1 - q) = 0.5$, $q = 0.5$	$\beta = 0.1$–50%

Table 3 Packet structures and drop rates

Packet drop rate	Packet structure			
	Data chunk number	Checksum chunk size	Data chunk size	Packet size
β	1	None	1, 468	1,500
$58\beta/1498$	1	10	1, 456	1,498
$70\beta/1494$	2	22	720	1,494
$76\beta/1500$	3	28	480	1,500
$82\beta/1490$	4	34	356	1,490
$8\beta/1492$	5	40	284	1,492

When the link is in the 'bad' state, a SCTP segment experiences a packet corruption in the network with the probability β. Table 2 summarizes the parameter values used for the error model.

To identify the corruption rate's thresholds as the criteria on whether or not to employ the checksum chunk option and on how many data chunks are contained in a data packet, we have first compared the standard SCTP with the proposed SCTP for the cases with 1, 2, 3, 4, and 5 data chunks per packet.

Notice that in the experiments the standard SCTP uses the fixed-size data chunk (1,468 bytes) and each packet carries only one data chunk. Thus every packet also has a fixed size of 1,500 bytes. Moreover, since the standard SCTP regards a corruption as a loss, all the corrupted packets are dropped by the receiver. Therefore, the packet drop rate is equal to β in the bad states for the standard SCTP.

On the contrary, the proposed scheme was evaluated with the variety of size of data chunks and packets. The detailed packet structure information is shown in Table 3. Also, in order to emulate the scenarios where the bit errors occur in the header portion of SCTP packets, the packet drop rate in the bad state is set to the proportion of the header size over the packet size. The detailed packet drop rates are also given in Table 3. Furthermore, in our simulation the number of the erased data chunks contained in a corrupted packet is randomly given.

The simulation results are shown in Fig. 8. In the figure, the horizontal axis denotes the packet corruption rate ranged from 0% to 50% (that is β value in the bad state of

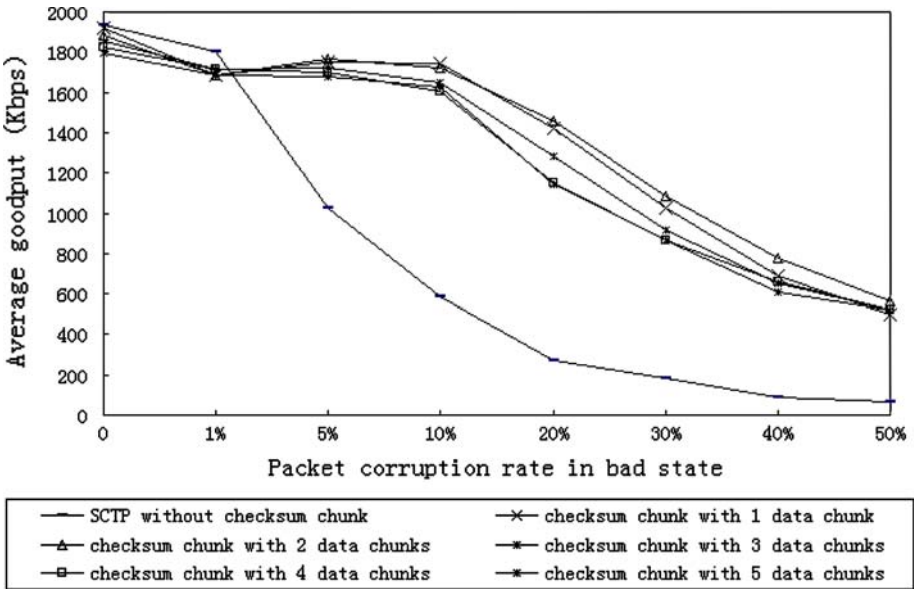

Fig. 8 Goodputs with different packet corruption rates

Gilbert model) and vertical axis indicates the average goodput that is calculated based on the cumulative TSN field contained in the final SACK chunk.

From the figure we can see that the standard SCTP without using checksum chunk gives higher goodputs than the cases of employing checksum chunk, only when the packet corruption rate (β value) is smaller than 1.5% roughly. This is because when the packet corruption rate is smaller, the performance gain of the proposed scheme cannot make up the amount of the overhead incurred by the checksum chunk scheme. In addition, the proposed scheme tends to keep the congestion window far larger than the existing scheme by differentiating a corruption event from the loss event. This may sometimes incur the buffer blocking problem at the receiver side. On the other hand, beyond that point, the proposed scheme appears to provide better throughput than the existing scheme. Such performance gain is anticipated since the proposed scheme does not decrease its *cwnd* and can recover as much available payload from the corrupted packet as possible, whereas the standard SCTP interprets all the corruption events as the network congestion and blindly halves its *cwnd* repeatedly.

In the figure we can also see that too many data chunks (e.g., more than three) may result in some undesirable side effects: (1) overhead caused by too huge checksum chunk as well as too many data chunks' headers, (2) much serious receiving buffer blocking problem and (3) raised packet loss rate incurred by header corruption, which will waste the more available bandwidth and decrease the transmission efficiency. Simulation results show that it is desirable to carry 1 or 2 chunks per packet, depending on the packet corruption rates. In particular, when the corruption rate in the bad state (β value) is larger than 15%, the checksum chunk strategy with two data chunks outperforms the strategy with one data chunk.

From the results, we can obtain the following observations: (1) we can identify 0.01 as the lower corruption rate's threshold, below which it is better not to use the checksum chunk option; (2) the higher corruption rate's threshold seems to be 0.1, above which the proposed checksum chunk scheme with two data chunks is preferred; and (3) if the sender want to

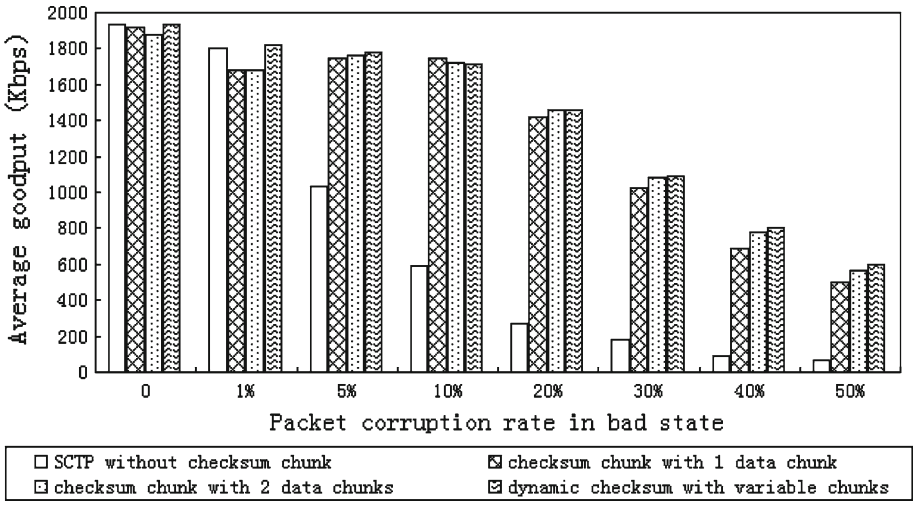

Fig. 9 Performance of SCTP with different chunk policies

apply the checksum chunk with a single data chunk, then the estimated corruption ratio lies between the two thresholds.

Figure 9 shows the simulation results in which we compare the goodput performance for different chunk policies. From the figure, we can see that the checksum chunk option with a flexible chunk policy outperforms the other approaches with a fixed number of data chunks. This is because the proposed scheme with a flexible chunk policy can overcome the potential problems of the checksum chunk scheme, as described above.

5 Conclusions

In this paper, we propose a partial CRC checksum chunk with flexible chunk policy to enhance the SCTP throughput performance under wireless environments. From the simulation results, we can see that the proposed checksum chunk could provide a significant throughput performance gain over the standard SCTP when the packet corruption rate is larger than 1%.

It is noted that the performance gain of the proposed scheme comes from the following features. First, the proposed scheme can recover the available data chunks from the corrupted packet and can deliver available in-order data to the upper layer so as to reduce the retransmission amount as much as possible. Second, by checking the corrupted TSN as well as corresponding timestamp enclosed in the responding SACK chunk, the proposed scheme can exploit a robust retransmission policy to avoid the unwanted timeouts. Third, the proposed scheme can distinguish the chunk corruptions from the chunk losses by using additional checksum chunk. Hence, it can avoid unnecessary deflation of the congestion window in the face of packet corruption.

Acknowledgements This research was supported by the MKE (Ministry of Knowledge Economy) of Korea, under the ITRC support program supervised by the IITA (IITA-2008-C1090-0801-0026). The authors also would like to sincerely thank R. Prasad, the editor in chief, and the anonymous reviewers for their careful reviews and useful suggestions.

References

1. Stewart, R., et al. (2000). Stream control transmission protocol. IETF, RFC 2960, October.
2. Fu, S., & Atiquzzaman, M. (2004). SCTP: State of the art in research, products, and technical challenges. *IEEE Communications Magazine, 42*(4), 64–76.
3. Grinnemo, K.-J., Andersson, T., & Brunstrom, A. (2005). Performance benefits of avoiding head-of-line blocking in SCTP. In *Proceedings of The Joint International Conference on Autonomic/Autonomous Systems (ICAS) International Conference on Networking and Services (ICNS)*. October 2005. Papeete, Tahiti, French Polynesia.
4. Ladha, S., Baucke, S., Ludwig, R., & Amer, P. D. (2004). On making SCTP robust to spurious retransmissions. *ACM SIGCOMM Computer Communication Review, 34*(2), 123–135.
5. Iyengar, J., et al. (2003). Making SCTP more robust to changeover. Presented at the International Symposium Performance Evaluation of Computer and Telecommunication Systems (SPECTS 2003).
6. Caro, A. L. Jr., et al. (2004). Retransmission schemes for end-to-end failover with transport layer multihoming. In *Proceedings of the IEEE Global Telecommunications Conference (GLOBECOM'04)*, November 2004, Vol. 3, pp. 1341–1347.
7. Iyengar, J. R., Amer, P. D., & Stewart, R. (2006). Concurrent multipath transfer using SCTP multihoming over independent end-to-end paths. *IEEE/ACM Transactions on Networking, 14*, 951–964.
8. Tonguz, O. K., & Yanmaz, E. (2003). On the theory of dynamic load balancing. In *Proceedings IEEE Global Telecommunications Conference (GLOBECOM'03)*, December 2003, Vol. 7, pp. 3626–3630.
9. Yanmaz, E., & Tonguz, O. K. (2005). Location dependent dynamic load balancing. In *Proceedings IEEE Global Telecommunications Conference (GLOBECOM'05)*, December 2005.
10. Abd El Al, A., Saadawi, T., & Lee, M. (2004). LS-SCTP: A bandwidth aggregation technique for stream control transmission protocol. *Computer Communications, 27*(10), 1012–1024.
11. Allman, M., et al. (1999). TCP congestion control. IETF, RFC 2581, April 1999.
12. Wang, D., Yang, S., & Sun, W. (2005). A Mac-error-warning method for SCTP congestion control over high BER wireless network. In *Wireless Communications, Networking and Mobile Computing, 2005. Proceedings. 2005 International Conference*, September 23–26, 2005, Vol. 1, pp. 513–516.
13. Lin, S., Costello, D. J., & Miller, M. J. (1984). Automatic-repeat-request errorcontrol schemes. *IEEE Communications Magazine, 22*, 5–17.
14. Ye, G., Saadawi, T. N., & Lee, M. J. (2004). Improving stream control transmission protocol performance over wireless networks. *Selected Areas in Communications, IEEE Journal, 22*(4), pp. 727–736.
15. Ramakrishnan, K., et al. (2001). The addition of explicit congestion notification (ECN) to IP. IETF, RFC 3168, September 2001.
16. Tickoo, O., Subramanian, V., Kalyanaraman, S., & Ramakrishnan, K. K. (2005). LT-TCP: End-to-end framework to improve TCP performance over networks with lossy channels. In *proceedings of IEEE 13th International Workshop on Quality of service (IWQoS)*, June 21–23, 2005, Passau.
17. Barakat, C., & Altman, E. (2002). Bandwidth tradeoff between TCP and link-level FEC. *Computer Networks, 39*(5), 133–150.
18. Barakat, C., & Fawal, A. A. (2004). Analysis of link-level hybrid FEC/ARQ-SR for wireless links and long-lived TCP traffic. *Performance Evaluation Journal, 57*(4), 423–500.
19. Baldantoni, L., Lundqvist, H., & Karlsson, G. (2004). Adaptive end-to-end FEC for improving TCP performance over wireless links. ICC 2004, June 2004.
20. Balan, R. K., Lee, B. P., Kumar, K. R. R., Jacob, L., Seah, W. K. G., Ananda, A. L. (2001). TCP HACK: TCP header checksum option to improve performance over lossy links. In *Proceedings IEEE INFOCOM 2001*, April 2001, Vol. 1, pp. 309–318.
21. Balakrishnan,H., & Katz, R. (1998). Explicit loss notification and wireless web performance. In *Proceedings IEEE Globecom Internet Mini-Conference*, November 1998.
22. Allman, M., Glover, D., & Sanchez, L. (1999). Enhancing TCP over satellite channels using standard mechanisms. RFC 2488, January 1999.
23. Liu, B., Goeckel, D., & Towsley, D. (2002). TCP-cognizant adaptive forward error correction in wireless networks. In *IEEE GLOBECOM'02*, November 17–21, 2002, Vol. 3, pp. 2128–2132.
24. Aguayo, D., Bicket, J., Biswas, S., Judd, G., & Morris, R. (2004). Link-level measurements from an 802.11b mesh network. SIGCOMM 2004, August 2004.
25. Network Simulator (ns-2), available from http://www.isi.edu/nsnam/ns/.
26. Gilbert, E. N. (1960). Capacity of a burst-noise channel. *Bell Systems Technical Journal, 39*, 1253–1265.

Author Biographies

Lin Cui received B.S. degree in Electronic Engineering from Tianjin University, China, and M.S. degree in Computer Engineering from Kyunghee University, Korea, in 1989 and 2005, respectively. He is now as a Ph.D. candidate with the Department of Computer Science in Kyungpook National University, Korea. His current research interests include Transport Layer Protocols, Wireless Communication and Internet Mobility.

Seok Joo Koh received B.S. and M.S. degrees in Management Science from KAIST in 1992 and 1994, respectively. He also received Ph.D. degree in Industrial Engineering from KAIST 1998. From August 1998 to February 2004, he worked for Protocol Engineering Center in ETRI. He is now an Associate Professor at Electrical Engineering and Computer Science in the Kyungpook National University since March 2004. His current research interests include Mobility Management for NGN, Internet Mobility and Transport Layer Protocols. He has so far participated in the International Standardization as an editor in ITU-T SG19, SG17, SG13, ISO/IEC JTC1/SC6 and IETF.